DEEP DIVE INTO DOCKER

From DevOps4Beginners

TABLE OF CONTENT

Table Of Contents: Deep Dive Into Docker

TABLE OF CONTENT

Table Of Contents: Deep Dive Into Docker

TABLE OF CONTENT

Table Of Contents: Deep Dive Into Docker

TABLE OF CONTENT

Table Of Contents: Deep Dive Into Docker

TABLE OF CONTENT

Table Of Contents: Deep Dive Into Docker

CHAPTER

Introduction to Course

COURSE INTRODUCTION

Course: Deep Dive Into Docker

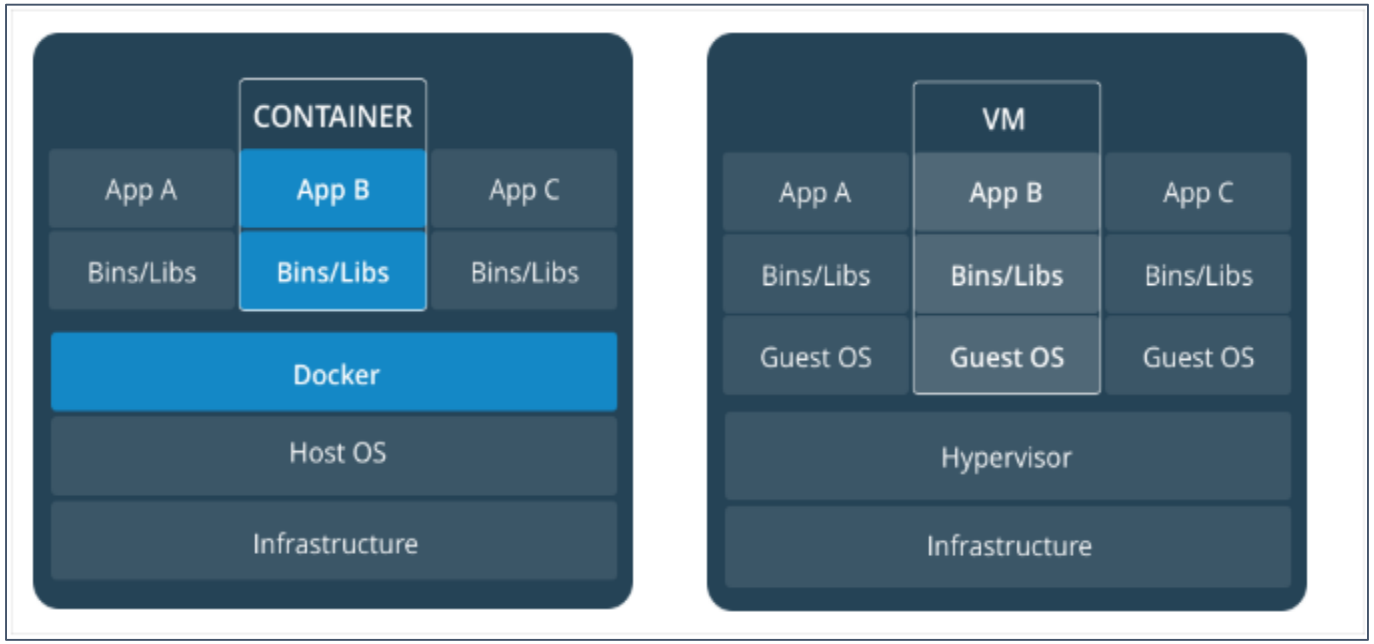
- Section 1: Introduction to Docker
- Section 2: Installing Docker CE on CentOS and Ubuntu
- Section 3: Docker Containers
- Section 4: Docker Images
- Section 5: Docker Storage
- Section 6: Docker Swarm / Orchestration
- Section 7: Docker Compose and Stack
- Section 8: Docker Networking
- Section 9: Docker Security
- Section 10: Other Topics

CHAPTER

Introduction to Docker

INTRODUCTION TO DOCKER

Introduction to Docker:



Docker is a tool that allows you to create, deploy, and run applications by using containers. Using docker you can run your software on different systems and environments like a development environment, a production environment. And, the software will run consistently, regardless of what kind of environment it's on.

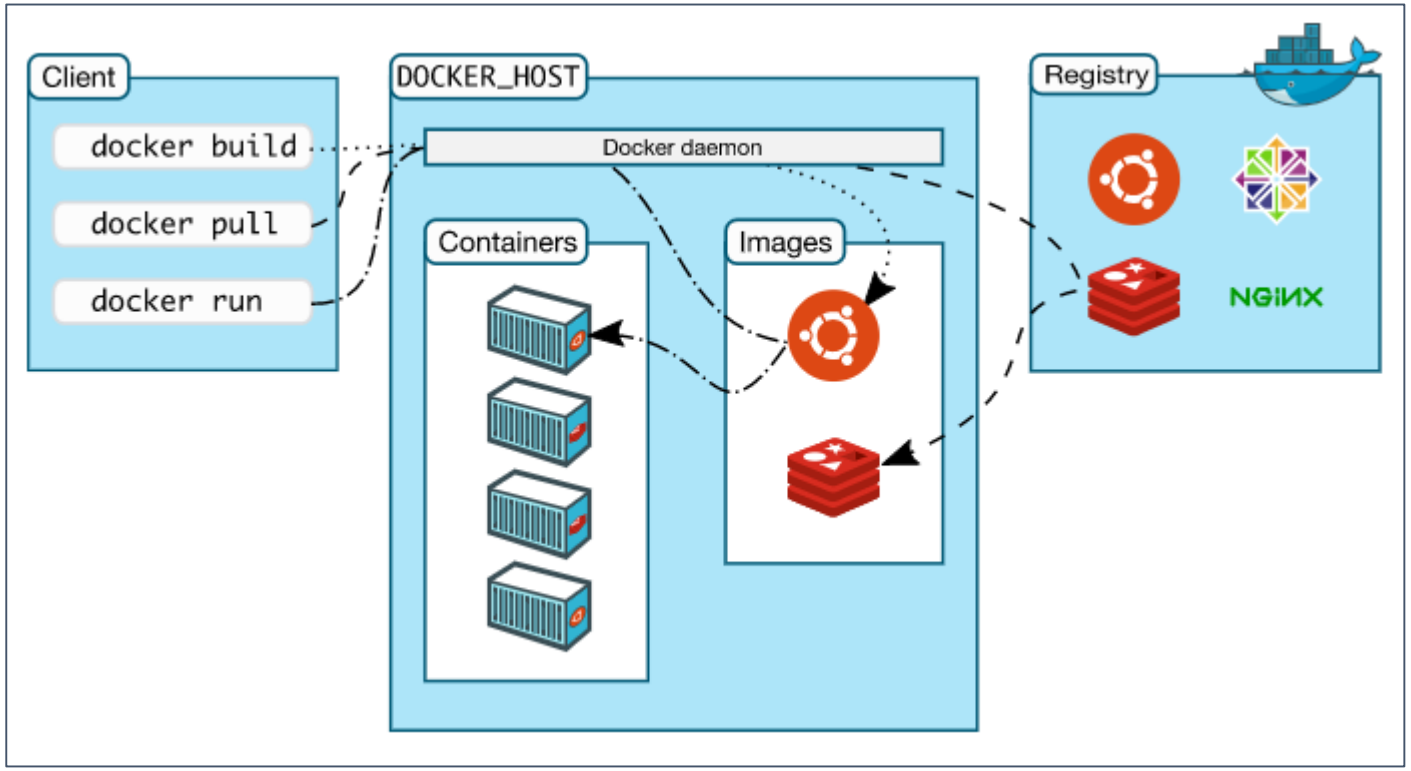
Reference Doc: https://docs.docker.com/get-started/#containers-and-virtual-machines

CHAPTER

Docker Architecture

DOCKER ARCHITECTURE

Docker Architecture:



Docker Client:
The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Docker Daemon (dockerd):
The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects

Docker Registries:
A Docker registry stores Docker images.

Reference Doc : https://docs.docker.com/get-started/overview/#docker-architecture

CHAPTER

Docker Installation

DOCKER INSTALLATION

Docker CE Installation Commands: **CentOS**

Step 1: Package Installation.

- sudo yum install -y yum-utils \
  device-mapper-persistent-data \
  lvm2

Step 2: Add Docker CE Repo.

- sudo yum-config-manager \
  --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo

Step 3: Install Docker CE packages

- sudo yum install docker-ce docker-ce-cli containerd.io

Reference Doc : https://docs.docker.com/install/linux/docker-ce/centos/

DOCKER INSTALLATION (CONT..)

Step 4: Start Docker Service.

- sudo systemctl start docker

Step 5: Enable Docker Service.

- sudo systemctl enable docker

Step 6: Check Docker Version.

- sudo docker version

Step 7: Add 'user' to 'docker' group.

- sudo usermod -a –G docker <whoami>

Step 8: Log-out & log-in. And, run "docker run" command.

- docker version
- docker run hello-world

Reference Doc : https://docs.docker.com/install/linux/docker-ce/centos/

DOCKER INSTALLATION

Docker CE Installation Commands: **Ubuntu**

Step 1: Package Installation.

- sudo apt-get update
- sudo apt-get -y install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common

Step 2: Add Docker GPG Key.

- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Reference Doc : https://docs.docker.com/install/linux/docker-ce/ubuntu/

DOCKER INSTALLATION (CONT..)

Step 3: Add repository.

- sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"

Step 4: Install Docker CE packages.

- sudo apt-get update
- sudo apt-get install docker-ce docker-ce-cli containerd.io

Step 5: Check Docker version.

- sudo docker version

Step 6: Add 'user' to 'docker' group.

- sudo usermod -a –G docker <whoami>

Step 7: Log-out & log-in. And, run command.

- docker version
- docker run hello-world

Reference Doc : https://docs.docker.com/install/linux/docker-ce/ubuntu/

RUNNING CONTAINER

Verify Installation.

If I run ‘docker version’ command before adding ‘user’ to ‘docker’ group I get permission denied error. Because user doesn’t have permission for ‘Docker Commands’. Hence, we have to give permission to user by adding user to ‘docker’ group to access docker commands.

- Error:

```
Client: Docker Engine - Community
 Version:           19.03.12
 API version:       1.40
 Go version:        go1.13.10
 Git commit:        48a66213fe
 Built:             Mon Jun 22 15:45:36 2020
 OS/Arch:           linux/amd64
 Experimental:      false
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock:
 /var/run/docker.sock: connect: permission denied
```

- Success:

```
Client: Docker Engine - Community
 Version:           19.03.12
 API version:       1.40
 Go version:        go1.13.10
 Git commit:        48a66213fe
 Built:             Mon Jun 22 15:45:36 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.12
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.10
  Git commit:       48a66213fe
  Built:            Mon Jun 22 15:44:07 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

CHAPTER

Docker Basic Commands: Part-1

BASIC COMMANDS – PART 1

Docker Basic Commands:

Instantiate a container using '**docker container run**' command and learn options and flags associated with it.

- **docker container run [OPTION1 OPTION2 … OPTIONn] [Image]:[TAG] [COMMAND] [ARGUMENT]**
  - **IMAGE**: Docker Image.
  - **TAG**: Run specific version of an image.
  - **COMMAND**: Command to run inside the container.
  - **ARGUMENT**: Arguments for the COMMAND.

Run Container:

- docker run hello-world
- docker **container** run hello-world (**Recommended way**)
- docker run nginx
- docker **container** run nginx (**Recommended way**)

Reference Doc : https://docs.docker.com/engine/reference/run/

BASIC COMMANDS – PART 1 (CONT..)

Run a container with COMMAND and ARGUMENT:

- docker run busybox **echo** Hello Students!
  - echo: Command run inside the busybox container.
  - Hello Students!: Argument for the Command.

List all containers (Running and stopped):

- docker ps -a
  - -a: All

Remove a stopped container:

- docker rm [Container ID]

CHAPTER

Docker Images and Containers

IMAGES AND CONTAINERS

Docker Image:

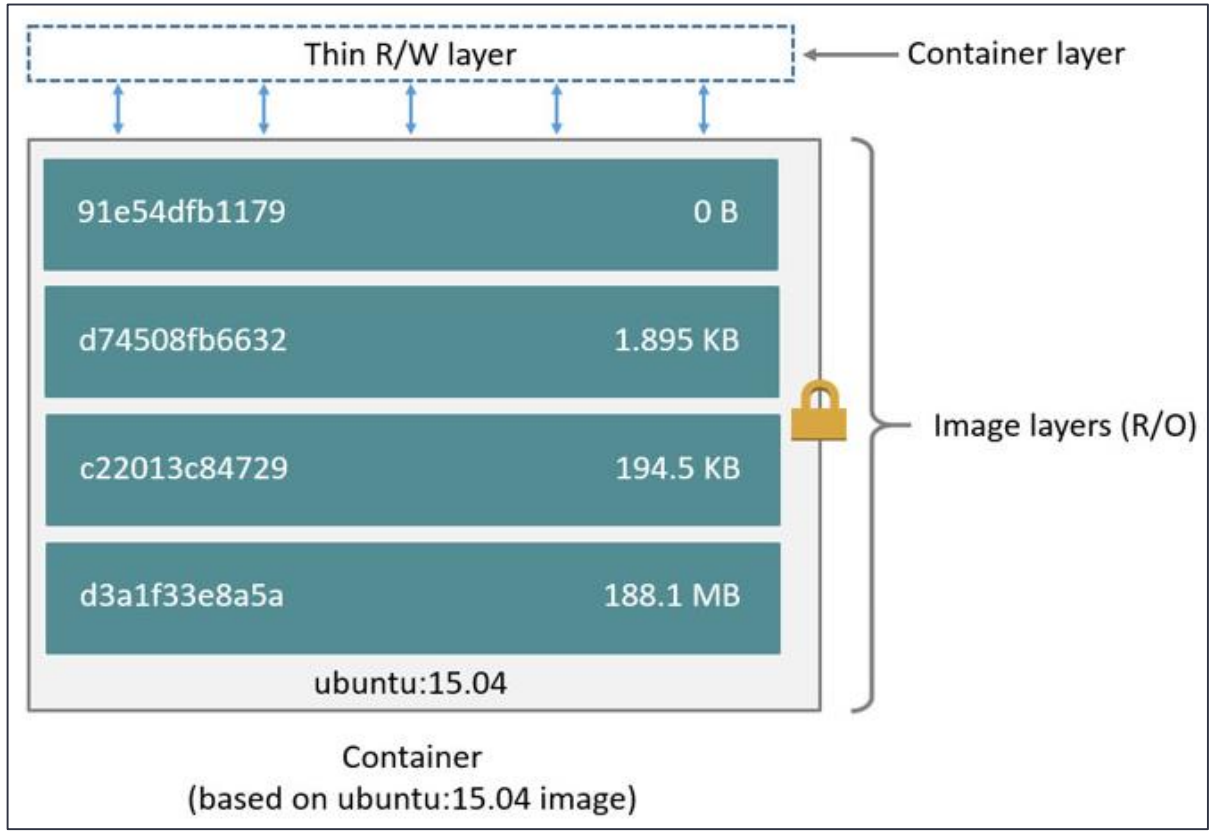


Image Source: https://docs.docker.com/storage/storagedriver/#images-and-layers

Image:
An image is built up of series of layers and each layer represents an instruction in the image.

Container layer:
When a container is created from an image it adds a new writable layer on top of the image layers.
This layer is called as "container layer".

The major difference between a container and an image is the top container layer.

IMAGES AND CONTAINERS (CONT..)

Multiple containers sharing the same image:

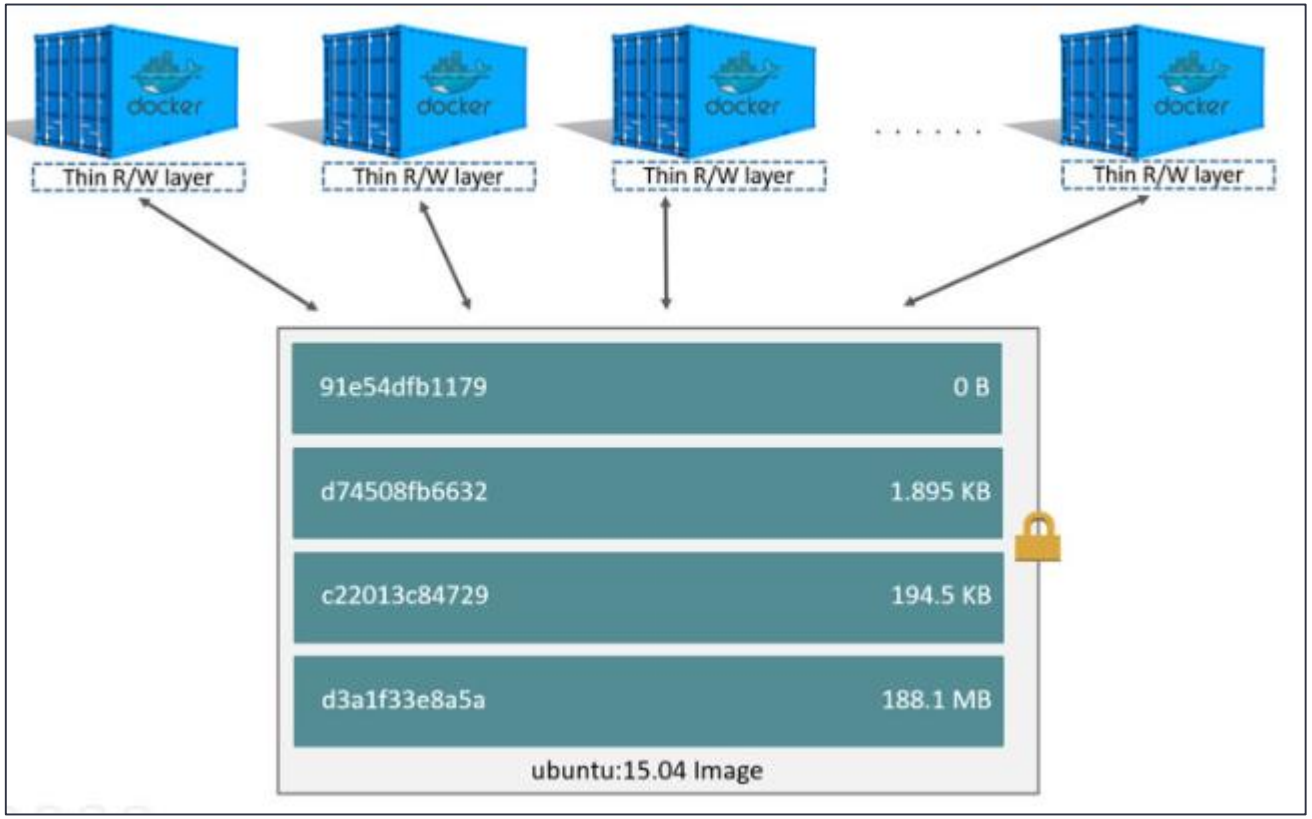


Image Source: https://docs.docker.com/storage/storagedriver/#container-and-layers

The diagram shows multiple containers sharing the same Ubuntu image. When you create containers from an image, the container and image become dependent on each other and you can't delete the image until all the containers attached to that image have been deleted.

When the container is deleted, the container layer is also deleted. However, the underlying image remains unchanged.

CHAPTER

Docker Basic Commands: Part-2

BASIC COMMANDS – PART 2

Docker help:

- docker **--help** | more
  - Management commands.
  - Commands.

Management Commands:

- **containers**: Manage containers
  - docker container --help
    - ✓ **run**: Run a command in a new container.
    - ✓ **ls**: List containers.
    - ✓ **rm**: Remove one or more containers.
- **image**: Manage images
- **network:** Manage networks
- **node**: Manage Swarm nodes

Reference Doc: https://docs.docker.com/engine/reference/commandline/container/

BASIC COMMANDS – PART 2 (CONT..)

Run a Container with options:

- docker container **run** nginx
- docker container run -d --name mynginx nginx:1.17.9
  - **-d (or) --detach**: Detached/Background Mode.
  - **--name**: Provide desired meaningful name.

List running containers:

- docker container **ls**

List all containers (Running and Stopped):

- docker container ls -a

Remove a container:

- docker container **rm** [container ID]

CHAPTER

Docker Basic Commands: Part-3

BASIC COMMANDS – PART 3

Publish Port(s):

There are 2 types:

- --publish (or) -p:
- --publish-all (or) -P:

--publish (or) -p:

Publish a container's port(s) to the host.

- docker container run -d --name [container name] **-p** [**Host port**]:[**Container port**] [Image]
  - Example:
    - ✓ docker container run -d --name mynginx -p 8080:80 nginx

--publish-all (or) -P:

Publish all exposed ports to random ports.

- docker container run -d --name [container name] **-P** [Image]
  - Example:
    - ✓ docker container run -d --name mynginx2 -P nginx

BASIC COMMANDS – PART 3 (CONT..)

Display detailed information of a container:

- docker container **inspect** [Container ID/Container name]
  - Example**:**
    - ✓ docker container inspect mynginx

List port mapping:

- docker container **port** [Container ID/Container name]
  - Example
    - ✓ Docker container port mynginx

Reference Doc:

https://docs.docker.com/engine/reference/run/

https://docs.docker.com/engine/reference/commandline/container/

CHAPTER

Docker Basic Commands: Part-4

BASIC COMMANDS – PART 4

--interactive (or) -i and --tty (or) -t:

When you detach from the container it's going to **stop** the container.

--interactive (or) -i: Keep STDIN open even if not attached

--tty (or) -t: Allocate a pseudo-TTY

- docker container run --name [container name] **-it** [Image]
  - Example
    - ✓ docker container run --name myubuntu -it ubuntu

attach:

Attach local standard input, output, and error streams to a **running** container.

When you detach from the container it's going to **stop** the container.

- docker container **attach** [Container name/Container ID]
  - Example
    - ✓ Docker container attach myubuntu

BASIC COMMANDS – PART 4 (CONT..)

exec:

Run a command in a **running** container.

exec will **not** stop the container when you detach from the running container.

- docker container **exec** [Options] [Container ID/Container name] [Command] [Arguments]
  - Example**:**
    - ✓ docker container exec -it myubuntu /bin/bash

CHAPTER

Docker Basic Commands: Part-5

BASIC COMMANDS – PART 5

Container Restart Policy:

Automatically start the containers when they exit, or when Docker restarts.

- docker container run [Options] **--restart** [restart policy] [Image]
- Types of restart policies:
  - no
  - on-failure
  - always
  - unless-stopped

no:

Default restart policy.

Do not automatically restart the container.

Example:

- docker container run --restart **no** nginx
- docker container run nginx (Same as above)

Reference Doc: https://docs.docker.com/config/containers/start-containers-automatically/

BASIC COMMANDS – PART 5 (CONT..)

on-filure:

Restart the container if it exits due to an error (i.e. non-zero exit code)

Example:

- docker container run --restart **on-failure** [Image]

always:

Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted.

Example:

- docker container run -d --name mynginxAlways --restart **always** -p 8080:80 nginx

unless-stopped:

Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.

Example:

- docker container run -d --name mynginxUnless --restart **unless-stopped** -p 8081:80 nginx

CHAPTER

Docker Basic Commands: Part-6

BASIC COMMANDS – PART 6

Container Basic Commands:

- **List running containers:**
  - docker container ls (Recommended way)
  - docker ps
- **List all containers (Running and Stopped):**
  - docker container ls -a (Recommended way)
  - docker ps -a
- **Stop a container:**
  - docker container **stop** [container ID/Container name]
- **Start a container:**
  - docker container **start** [container ID/Container name]
- **Pause a container:**
  - docker container **pause** [container ID/Container name]
- **Unpause a container:**
  - docker container **unpause** [container ID/Container name]

BASIC COMMANDS – PART 6 (CONT..)

- **Fetch the logs of a container:**
  - docker container **logs** [Container name/Container ID]
- **To see container resource usage statistics**
  - docker container **stats** [Container name/Container ID]
- **To see running processes of a container:**
  - docker container **top** [container ID/Container name]

Image Basic Commands:

- **Pull an image:**
  - docker image **pull** [Image]
- **List images:**
  - docker image **ls**
- **To see detailed information of an image:**
  - docker image **inspect** [Image]

BASIC COMMANDS – PART 6 (CONT..)

Clean Up: Remove Images and Containers.

- **Remove a stopped container:**
  - docker container **rm** [Container Name/Container ID]
- **Remove all stopped containers:**
  - docker container **prune**
- **Remove a running container :**
  - docker container **rm -f** [Container Name/Container ID]
- **Remove all stopped and running containers :**
  - docker container rm -f **`docker ps -a -q`**
  - docker container rm –f **`docker container ls -a -q`**
- **Remove an image:**
  - docker image **rm** [Image]
- **Automatically remove a container when it exits:**
  - docker container run **--rm** [Image]

CHAPTER

Uninstall & Upgrade Docker Engine

UNINSTALL & UPGRADE DOCKER ENGINE

Uninstall Docker Engine:

- sudo systemctl stop docker
- sudo apt-get remove -y docker-ce docker-ce-cli
- sudo apt-get update

Install Docker Engine (Lower Version):

- sudo apt-get install -y docker-ce=5:18.09.4~3-0~ubuntu-bionic docker-ce-cli=5:18.09.4~3-0~ubuntu-bionic

Check Docker Engine Version:

- docker version

Reference Doc:

https://docs.docker.com/engine/install/ubuntu

https://docs.docker.com/engine/install/ubuntu/#uninstall-old-versions

UNINSTALL & UPGRADE DOCKER ENGINE (CONTD..)

Upgrade Docker Engine:

- sudo apt-get install -y docker-ce=5:18.09.5~3-0~ubuntu-bionic docker-ce-cli=5:18.09.5~3-0~ubuntu-bionic

Check Docker Engine Version:

- docker version

Reference Doc:

https://docs.docker.com/engine/install/ubuntu/#upgrade-docker-engine

CHAPTER -6

Docker Swarm

DOCKER SWARM

Docker Swarm:

- Run containers on multiple servers as a cluster.
- Build distributed cluster of Docker machine.
- Supports orchestration, high-availability, Scaling, load balancing etc..

Manager:

- Assign work to worker nodes.
- Responsible for controlling the cluster and orchestration.

Workers:

- Responsible for running container workloads.

Reference Doc : https://docs.docker.com/engine/swarm/

DOCKER SWARM (CONTD..)

Configure Swarm Manager:

- Install Docker CE. (Section 3: Chapter – 1/2).
- docker info | grep swarm
- docker swarm init --advertise-addr [**Swarm Manager Private IP**]
- docker info | grep swarm
- docker node ls

Reference Doc:

https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/

DOCKER SWARM (CONTD..)

Add worker Node to Swarm Manager:

- Install Docker CE. (Section 3: Chapter 1/2).
- docker swarm join-token worker (On Swarm Manager)
- Copy and run the **swarm join-token** output. (On Worker Node).
- docker node ls (On Swarm Manager)

Reference Doc:

https://docs.docker.com/engine/swarm/swarm-tutorial/add-nodes/

CHAPTER

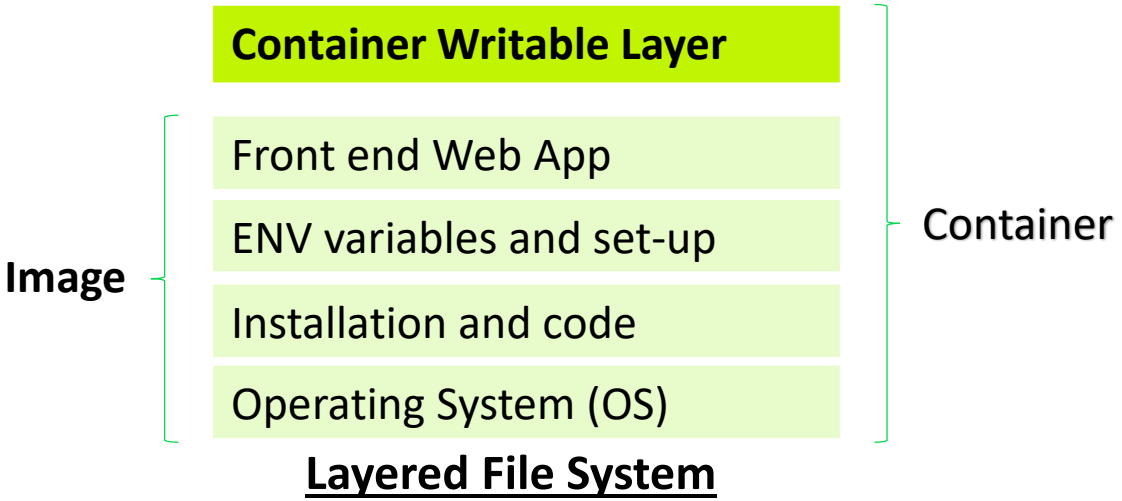
Docker Images

DOCKER IMAGES

IMAGES:

Docker image is a file which contains dependencies, binaries and required configurations to run software inside a container.

- docker pull [**Image Name**]:[tag]
- docker image pull [**Image Name**]:[tag] (**Recommended way**)



Reference Doc : https://docs.docker.com/storage/storagedriver/

DOCKER IMAGES (CONT..)

List all layers of an image:

- docker image history nginx

```
IMAGE          CREATED        CREATED BY
ed21b7a8aee9   2 weeks ago    /bin/sh -c #(nop) 11 CMD ["nginx" "-g" "daemon…
<missing>      2 weeks ago    /bin/sh -c #(nop) 10 STOPSIGNAL SIGTERM
<missing>      2 weeks ago    /bin/sh -c #(nop) 9 EXPOSE 80
<missing>      2 weeks ago    /bin/sh -c ln -sf 8 /dev/stdout /var/log/nginx…
<missing>      2 weeks ago    /bin/sh -c set -x 7     && addgroup --system -…
<missing>      2 weeks ago    /bin/sh -c #(nop) 6 ENV PKG_RELEASE=1~buster
<missing>      2 weeks ago    /bin/sh -c #(nop) 5 ENV NJS_VERSION=0.3.9
<missing>      2 weeks ago    /bin/sh -c #(nop) 4 ENV NGINX_VERSION=1.17.9
<missing>      2 weeks ago    /bin/sh -c #(nop) 3 LABEL maintainer=NGINX Do…
<missing>      2 weeks ago    /bin/sh -c #(nop) 2 CMD ["bash"]
<missing>      2 weeks ago    /bin/sh -c #(nop) 1 ADD file:d1f1b387a158136fb…
```

IMAGE HISTORY

```
1   ADD file ... in /
2   CMD ["bash"]
3   LABEL maintainer=NGINX Docker Maintainers
4   ENV NGINX_VERSION=1.17.9
5   ENV NJS_VERSION=0.3.9
6   ENV PKG_RELEASE=1~buster
7   /bin/sh -c set -x
8   /bin/sh -c ln -sf /dev/stdout
9   EXPOSE 80
10  STOPSIGNAL SIGTERM
11  CMD ["nginx" "-g" "daemon
```

Reference Doc : Link to nginx image history and nginx dockerfile

CHAPTER

Dockerfile-Part 1

DOCKERFILE – PART 1

Dockerfile:

Dockerfile is a set of instructions and commands used to build an image.

Build Image:

- docker image build -t [TAG] .
- docker image build -t [TAG] -f [Dockerfile Name] .
- docker image build --no-cache -t [TAG] .

```
## Dockerfile Stracture
FROM [Image]:[tag]
LABEL [Key]=[value]
RUN [command]
```

```
## My First Dockerfile
FROM ubuntu:18.04
LABEL version="v1.0"
LABEL description="Building my first image"
RUN apt-get update -y
```

Reference Doc : https://docs.docker.com/engine/reference/builder/

DOCKERFILE – PART 1 (CONT..)

Key Points To Remember:

- Ephemeral container.
- Order of execution.
- Keep image size minimum.
  - Avoid unnecessary packages and files.
  - Use multi-stage build.
  - Keep number of layers to minimum.

Building an Image:

```
[root@Devops4Beginners ~]# cat Dockerfile
FROM ubuntu:18.04
LABEL version="v1.0"
LABEL description="Building my first image"
RUN apt-get update -y
[root@Devops4Beginners ~]# docker image build -t myfirstimage:v1 .
Sending build context to Docker daemon   47.1kB
Step 1/4 : FROM ubuntu:18.04
 ---> c3c304cb4f22
Step 2/4 : LABEL version="v1.0"
 ---> Using cache
 ---> 863fc69b667a
Step 3/4 : LABEL description="Building my first image"
 ---> Using cache
 ---> e64671822c6c
Step 4/4 : RUN apt-get update -y
 ---> Using cache
 ---> 85b76896eff4
Successfully built 85b76896eff4
Successfully tagged myfirstimage:v1
```

Reference Doc:

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

CHAPTER

Dockerfile Part -2

DOCKERFILE – PART 2 (CONT..)

Frequently used Dockerfile Instructions:

- **FROM**
  - Sets base/parent Image.
- **LABEL**
  - Adds metadata to the image.
- **RUN**
  - Creates new layer.
- **EXPOSE**
  - Intend port to publish.
- **CMD**
  - Setting default command for container. It can be overridden.
- **ENTRYPOINT**
  - Specify executable inside the container. It does **not** get overridden.
  - However, it can be overridden by --**entrypoint** flag.

Reference Doc : https://docs.docker.com/engine/reference/builder/

DOCKERFILE – PART 2 (CONT..)

Sample Dockerfile:

```
## Sample Dockerfile for nginx.
FROM ubuntu

LABEL description = "My image for nginx Web Server"
LABEL version = "V1.0"

RUN apt-get update -y
RUN apt-get install curl -y
RUN apt-get install nginx -y

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

```
RUN apt-get update -y && \
    apt-get install curl -y && \
    apt-get install nginx -y
```

Combining RUN instructions into one line.

CHAPTER

Dockerfile Part -3

DOCKERFILE (CONT..)

Frequently used Dockerfile Instructions (cont..):

- **WORKDIR**
  - Sets current working directory.
- **COPY**
  - Copy file from one location to container.
  - If spaces include quotes
- **ADD**
  - Similar to ADD instruction with additional features.
  - if spaces include quotes.
  - Download a file from URL.
    - ✓ **ADD** http://<www.abcxyz.com>/downloads/file.zip

Reference Doc : https://docs.docker.com/engine/reference/builder/

DOCKERFILE (CONT..)

Sample Dockerfile:

```
1   FROM ubuntu
2
3   LABEL description = "My image for nginx Web Server"
4   LABEL version = "V1.0"
5
6   RUN apt-get update -y && \
7       apt-get install curl -y && \
8       apt-get install nginx -y
9
10  WORKDIR /var/www/html
11  COPY index.html ./
12
13  EXPOSE 80
14
15  CMD  ["nginx", "-g", "daemon off;"]
```

```
10  WORKDIR /var/www
11  WORKDIR html
```

In the above snapshot, html does not start with / (slash) so it becomes **relative** to /var/www.
Which is same as WORKDIR /var/www/html

CHAPTER

Dockerfile Part -4

DOCKERFILE (CONT..)

Frequently used Dockerfile Instructions (cont..):

- **ENV**
  - Set environment variables.
  - Can be overridden by **--env** flag.
  - ENV [**Key**]=[**Value**]
- **USER**
  - Set user.

```
RUN useradd -ms /bin/bash devopsuser
USER devopsuser
WORKDIR /home/devopsuser
```

Reference Doc : https://docs.docker.com/engine/reference/builder/

DOCKERFILE (CONT..)

Sample Dockerfile: ENV and USER Instructions.

```
FROM ubuntu

LABEL description="My image for nginx Web Server"
LABEL version="V1.0"

ENV PORT="80"
ENV ENVIRONMENT="development"

RUN apt-get update -y && \
    apt-get install curl -y

RUN useradd -ms /bin/bash devopsuser
USER devopsuser
WORKDIR /home/devopsuser
```

```
[root@Devops4Beginners ~]# #vi Dockerfile
[root@Devops4Beginners ~]# #docker image build -t myubuntu .
[root@Devops4Beginners ~]# #docker container run -d -it --name myUbuntuUser  myubuntu
```

```
[root@Devops4Beginners ~]# docker container exec -it myUbuntuUser /bin/bash
devopsuser@1179b62a016a:~$ whoami
devopsuser
devopsuser@1179b62a016a:~$ pwd
/home/devopsuser
devopsuser@1179b62a016a:~$
```

CHAPTER

Dockerfile Part -5

DOCKERFILE (CONT..)

Frequently used Dockerfile Instructions (cont..):

- **HEALTHCHECK**
  - Checks the health of a container by running a command inside the container.
  - Can be only one Healthcheck instruction in a Dockerfile.
  - **Options for CMD:**
    - ✓ --interval=DURATION (default: 30s)
    - ✓ --timeout=DURATION (default: 30s)
    - ✓ --start-period=DURATION (default: 0s)
    - ✓ --retries=N (default: 3)

      HEALTHCHECK --interval=5s CMD curl localhost:<port>

- **ARG**
  - Declared before the FROM instruction.

Reference Doc : https://docs.docker.com/engine/reference/builder/

DOCKERFILE (CONT..)

Sample Dockerfile: HEALTHCHECK and ARG Intructions.

```
ARG VERSION="19.10"
FROM ubuntu:$VERSION

LABEL description="My image for nginx Web Server"
LABEL version="V1.0"

ENV PORT="80"
ENV ENVIRONMENT="development"

RUN apt-get update -y && \
    apt-get install curl -y && \
    apt-get install nginx -y

WORKDIR /var/www/html
COPY index.html ./

EXPOSE $PORT

CMD  ["nginx", "-g", "daemon off;"]

HEALTHCHECK --interval=5s CMD curl localhost:80
```

CHAPTER

Docker Image CLI

DOCKER IMAGE CLI (CONT..)

- **Pull an image:**
  - docker image pull nginx
  - docker image ls
- **Search an Image:**
  - docker search nginx
- **Limit the number of result:**
  - docker search --limit 10 nginx
- **Filter search result:**
  - docker search --filter stars=200 nginx
  - docker search -f stars=100 -f is-official=true nginx

Reference Doc:

https://docs.docker.com/engine/reference/commandline/image/

https://docs.docker.com/engine/reference/commandline/docker/

DOCKER IMAGE CLI (CONT..)

- **List images:**
  - docker images
  - Docker image ls
  - Docker image ls –a
- **Tag an image:**
  - docker image tag [Source Image]:[tag] [Reference to source image]:[tag]
    - ✓ **docker tag ubuntu myubuntu:v1**
- **Delete an image:**
  - docker image rm nginx
  - docker rmi nginx

Reference Doc:

https://docs.docker.com/engine/reference/commandline/image/

DOCKER IMAGE CLI (CONT..)

- **Remove dangling image:**
  - docker image prune
- **Remove all unused and dangling image:**
  - docker image prune -a
- **Inspect an image:**
  - docker image inspect nginx
  - docker image inspect nginx - -format”{{.ContainerConfig.Hostname}}”

Reference Doc:

https://docs.docker.com/engine/reference/commandline/image/

CHAPTER

Flattening an Image

FLATTENING AN IMAGE

Execution Steps:

Flattening an image to a single layer to save some space and get an extra performance.

Flattening an Image:

- docker **export**
- docker **import**
- docker image **history**

- **Before flattening:**

```
[root@Devops4Beginners ~]# docker image history multilayer
IMAGE          CREATED          CREATED BY                                      SIZE
28403985cf62   About a minute ago   /bin/sh -c apt-get install curl -y          16.2MB
c37588734dc5   About a minute ago   /bin/sh -c apt-get update -y                22.1MB
e13c501b0de2   2 minutes ago    /bin/sh -c #(nop)  LABEL version== V1.0         0B
ccf13a78abc3   2 minutes ago    /bin/sh -c #(nop)  LABEL description== My cu…   0B
74435f89ab78   13 days ago      /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      13 days ago      /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>      13 days ago      /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      13 days ago      /bin/sh -c [ -z "$(apt-get indextargets)" ]     1.01MB
<missing>      13 days ago      /bin/sh -c #(nop) ADD file:b2342c7e6665d5ff3…   72.8MB
```

- **After flattening:**

```
[root@Devops4Beginners ~]# docker image history newmyubuntu
IMAGE          CREATED          CREATED BY       SIZE          COMMENT
daf1defe2b29   About a minute ago                110MB         Imported from -
```

CHAPTER

Multi Stage Builds

MULTI-STAGE BUILDS

Multi-Stage Builds:

- Multi-stage builds will have more than one FROM instructions in the Dockerfile.
- Each FROM instruction creates a new build.

```
# Multi-Stage Builds Example
FROM golang:1.14.2
WORKDIR /gopgm
COPY hellostudents.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o hellostudents .

FROM alpine:3.11.6 AS os
WORKDIR /root
COPY --from=0 /gopgm/hellostudents .

FROM os
CMD ["./hellostudents"]
```

Reference Doc : https://docs.docker.com/develop/develop-images/multistage-build/

CHAPTER

Save and Load an Image

SAVE AND LOAD AN IMAGE

Save an Image:

Save one or more images to a tar archive.

- docker image **save** [image name] > [archive name].tar

Load an Image:

Load an image from a tar archive or STDIN.

- docker image **load** < [archive name]

Reference Doc :

https://docs.docker.com/engine/reference/commandline/save/

https://docs.docker.com/engine/reference/commandline/load/

CHAPTER

Persistent and Non-persistent Storage

PERSISTENT AND NON-PERSISTENT STORAGE

Storage Driver:

Provides temporary internal storage for containers.

Manages and controls how images and containers are stored on your Docker host.

Linux distribution	Recommended storage drivers	Alternative drivers
Docker Engine - Community on Ubuntu	`overlay2` or `aufs` (for Ubuntu 14.04 running on kernel 3.13)	`overlay` [1], `devicemapper` [2], `zfs`, `vfs`
Docker Engine - Community on Debian	`overlay2` (Debian Stretch), `aufs` or `devicemapper` (older versions)	`overlay` [1], `vfs`
Docker Engine - Community on CentOS	`overlay2`	`overlay` [1], `devicemapper` [2], `zfs`, `vfs`
Docker Engine - Community on Fedora	`overlay2`	`overlay` [1], `devicemapper` [2], `zfs`, `vfs`

Reference Doc : https://docs.docker.com/storage/storagedriver/select-storage-driver/
https://success.docker.com/article/compatibility-matrix

PERSISTENT AND NON-PERSISTENT STORAGE

Docker Storage:

Store and manage container data.

Two types of storage:

1. Non-Persistent
2. Persistent

Non-Persistent Storage:

- Data resides within the container
- Get deleted when container deleted
- All container has it by default.
- Storage Drivers:
  - RHEL/Latest Ubuntu & CentOS uses Overlay2
  - Ubuntu 14 and older uses aufs
  - CentOS 7 and older uses devicemapper
  - Windows uses its own.
- Storage Location:
  - Linux: /var/lib/docker/[STORAGE-DRIVER]/
  - Windows: C:\ProgramData\Docker\windowsfilter\

PERSISTENT AND NON-PERSISTENT STORAGE (CONT..)

Persistent Storage:

- Data does not reside within the container
- Does not get deleted when container deleted
- **Two types Persistent Storage:**
  1. Volumes:
     - Mounted to a directory in a container.
     - Storage Location:
       - ✓ Linux: /var/lib/docker/volumes/
       - ✓ Windows: C:\ProgramData\Docker\volumes
     - Supports 3rd party drivers:
       - ✓ Block Storage e.g. Amazon AWS EBS.
       - ✓ File Storage e.g. Amazon AWS EFS.
       - ✓ Object Storage e.g. Amazon AWS S3.
  2. Bind Mounts:
     - File or directory on the host system is mounted into a container's file or directory.

Reference Doc : https://docs.docker.com/storage/

CHAPTER

Docker Storage-Volumes

DOCKER STORAGE - VOLUMES

Docker Storage – Volumes:

Mounted to a directory in a container.

Volume CLI:

- Create a Volume.
  - docker volume create [**volume name**]
- List Volumes.
  - docker volume ls
- Inspect a Volume.
  - docker volume inspect [**volume name**]
- Remove a volume.
  - docker volume rm [**volume name**]
- Delete all unused volumes.
  - docker volume prune

Reference Doc: https://docs.docker.com/storage/volumes/

DOCKER STORAGE - VOLUMES (CONT..)

Two ways to mount volume into a container:

1. **- - mount**

   Syntax:

   ```
   docker container run -d \
           --name mynginx1 \
           --mount type=volume,\
             source=nginxvolume,\
             target=/usr/share/nginx/html/ \
             nginx
   ```

   docker volume create nginxvolume

   directory inside mynginx1 container

2. **- - volume or - v**

   Syntax:

   ```
   docker container run -d \
           --name mynginx2 \
           -v nginxvolume:/usr/shared/nginx/html/ \
           nginx
   ```

   Volume name should NOT start with slash (/)
   Correct: -v nginxvolume:/usr/shared/nginx/html/
   Wrong: -v /nginxvolume:/usr/shared/nginx/html/

Reference Doc : https://docs.docker.com/storage/volumes/

CHAPTER

Docker Storage-
Bind Mounts

DOCKER STORAGE – BIND MOUNTS

Docker Storage – Bind Mounts:

File or directory on the host system is mounted into a container's file or directory.

Two ways to create Bind Mounts:

1. **- - mount**

Syntax:

```
docker container run -d \
    --name nginxbind1 \
    --mount type=bind,\
      source="$(pwd)"/bindexample,\
      target=/app \
      nginx
```

mkdir bindexample

directory inside nginxbind1 container

2. **- - volume or –v**

Syntax:

```
docker container run -d \
    --name nginxbind2 \
    -v /user/username/bindexample2:/app \
    nginx
```

Bind Mount is a file or directory on the host system.
Therefore, Bind Mount name should start with slash (/)
Correct: -v /user/username/bindexample2:/app
Wrong: -v user/username/bindexample2:/app

Reference Doc : https://docs.docker.com/storage/bind-mounts/

CHAPTER

Dockerfile-
Volume Instruction

DOCKERFILE – VOLUME INSTRUCTION

Volume Instruction:

Volume instruction automatically creates a volume and mounts that volume to specified directory.

Dockerfile:

```
## Sample Dockerfile
FROM nginx
LABEL description="Using Volume Instruction"
VOLUME ["/usr/share/nginx/html/"]
```

Reference Doc : https://docs.docker.com/engine/reference/builder/#volume

CHAPTER

Storage Driver

STORAGE DRIVER

Storage Driver:

- Provides temporary internal storage for containers.
- Manages and controls how images and containers are stored on your Docker host.

Linux distribution	Recommended storage drivers	Alternative drivers
Docker Engine - Community on Ubuntu	`overlay2` or `aufs` (for Ubuntu 14.04 running on kernel 3.13)	`overlay` [1], `devicemapper` [2], `zfs`, `vfs`
Docker Engine - Community on Debian	`overlay2` (Debian Stretch), `aufs` or `devicemapper` (older versions)	`overlay` [1], `vfs`
Docker Engine - Community on CentOS	`overlay2`	`overlay` [1], `devicemapper` [2], `zfs`, `vfs`
Docker Engine - Community on Fedora	`overlay2`	`overlay` [1], `devicemapper` [2], `zfs`, `vfs`

Reference Doc : https://docs.docker.com/storage/storagedriver/select-storage-driver/
https://success.docker.com/article/compatibility-matrix

STORAGE DRIVER (CONT..)

Check default Storage driver:

- docker info
- docker info | grep storage

Method -1 : Edit unit file (docker.service)

- Add **--storage-driver** flag
  - sudo vi /lib/systemd/system/docker.service
  - ExecStart=/usr/bin/dockerd **--storage-driver** devicemapper -H fd:// --containerd=/run/containerd/containerd.sock

- Restart the docker
  - sudo systemctl daemon-reload
  - sudo systemctl restart docker

STORAGE DRIVER (CONT..)

<u>Method 2</u>: Configuration file (daemon.json)

- Configure daemon file
  - sudo vi /etc/docker/daemon.json

```
1  {
2      "storage-driver":"devicemapper"
3  }
```

- Restart Docker
  - sudo systemctl restart docker
  - sudo systemctl status docker

CHAPTER

Introduction to Docker Swarm

INTRODUCTION TO DOCKER SWARM

Docker Swarm:

- Build distributed cluster of Docker machine. Cluster consists of one or more nodes.
- Run containers on multiple servers as a cluster.
- Supports orchestration, high-availability, Scaling, load balancing, rolling updates, rollbacks etc..
- Swarm uses mutual Transport Layer Security (TLS) for communication and authentication of nodes.

Two Types of Node in Swarm:

1. **Manager**
   - Assign work to worker nodes.
   - Responsible for controlling the cluster and orchestration.

2. **Worker**
   - Accepting tasks from the Manager node and running container workloads.

Reference Doc : https://docs.docker.com/engine/swarm/

INTRODUCTION TO DOCKER SWARM (CONT..)

Docker Swarm Cluster:

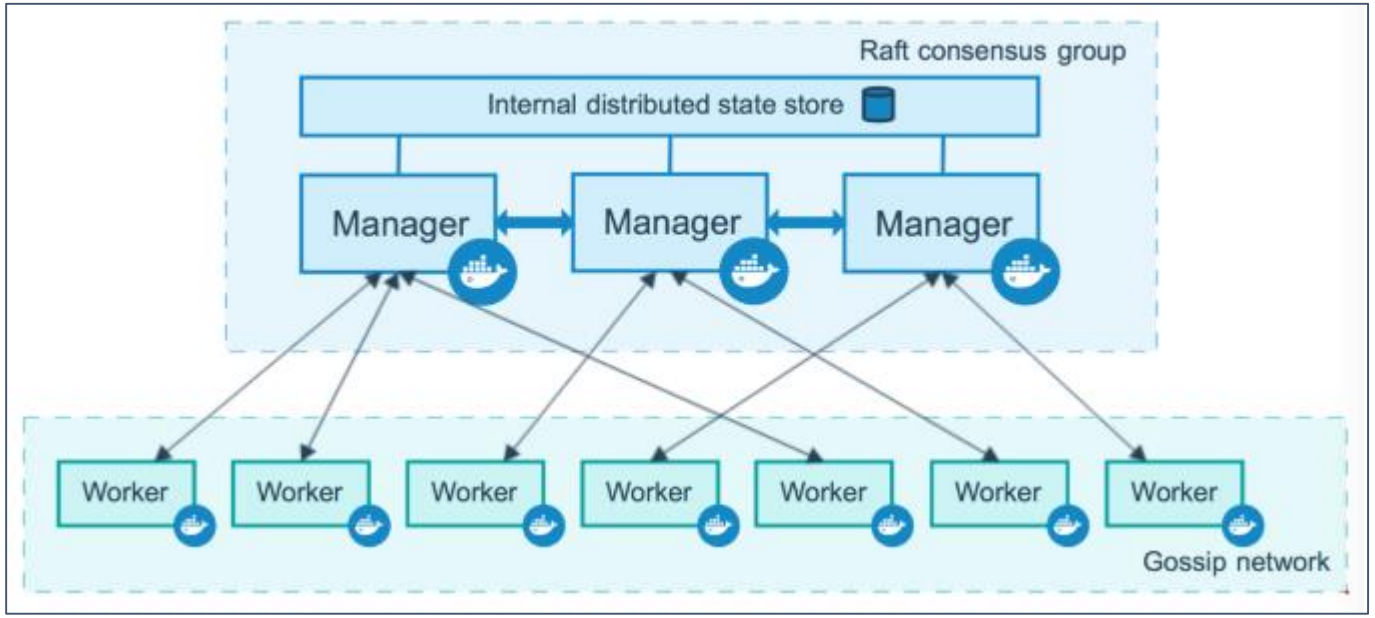


Image Source: https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

Manager/s assign work to Worker node/s. And, Swarm uses mutual Transport Layer Security (TLS) for communication.

CHAPTER

Docker Swarm Configuration

DOCKER SWARM CONFIGURATION (CONT..)

Docker Swarm Set-up:

1. Configure Swarm Manager.
2. Add worker node to Swarm manager.

Configure Swarm Manager:

- Install Docker CE. (Section 3: Chapter – 1/2).
- docker info | grep Swarm
- docker swarm init --advertise-addr [**Node Private IP**]
- docker info | grep Swarm
- docker node ls

Reference Doc:

https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/

DOCKER SWARM CONFIGURATION (CONT..)

Add worker Node to Swarm Manager:

- Install Docker CE. (Section 3: Chapter – 1/2).
- docker swarm join-token worker (On Swarm Manager)
- Copy and run the **swarm join-token** output (On Worker Node)
- docker node ls (On Swarm Manager)

Reference Doc:

https://docs.docker.com/engine/swarm/swarm-tutorial/add-nodes/

CHAPTER

Docker Swarm and Node Commands

DOCKER SWARM AND NODE COMMANDS

Swarm and Node Commands:

- List all nodes. (On Manager)
  - docker node ls
- To inspect a node
  - docker node inspect [**Node Id**]
- Promote a node to Manager.
  - docker node promote [**Node Id**]
- Demote a node to Worker
  - docker node demote [**Node Id**]
- Remove a node from Swarm

  Step1: On Manager
  - docker node rm -f [**Node name**]

  Step 2: On Worker
  - docker swarm leave

Reference Doc: https://docs.docker.com/engine/reference/commandline/node/

DOCKER SWARM AND NODE COMMANDS (CONT..)

- Generate Join-token for worker. (On Manager).
  - docker swarm join-token **worker**
- Generate join-token for manager. (On Manager).
  - docker swarm join-token **manager**

Reference Doc:

https://docs.docker.com/engine/reference/commandline/swarm/

CHAPTER

Docker Swarm Autolock

DOCKER SWARM AUTOLOCK

Docker Swarm:

- Encrypts RAFT logs and TLS communication between nodes.

Docker Swarm Autolock:

- Provides an un-lock key to un-lock Swarm whenever docker restart.

Commands:

- **Turn on Autolock**
  - docker swarm init --autolock=true
  - docker swarm update --autolock=true.
- **Turn off Autolock.**
  - docker swarm update --autolock=false
- **Unlock Swarm manager**
  - docker swarm unlock
- **Retrieve unlock key**
  - docker swarm unlock-key
- **Rotate unlock key**
  - docker swarm unlock-key --rotate

Reference Doc: https://docs.docker.com/engine/swarm/swarm_manager_locking/

CHAPTER

Introduction to Docker Services

INTRODUCTION TO DOCKER SERVICES

Docker Service:

- Allow us to run applications in the Swarm cluster.
- One or more containers can be run across the nodes in Swarm cluster.

Difference:

docker container run	docker service create
Runs a single container on a single host	Runs container(s) on 1 to n nodes
Not highly available	Highly available
Not easily scalable	Easily scalable (up or down)
Can’t use -- replicas flag	--replicas used to scale.

Reference Doc :

https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

INTRODUCTION TO DOCKER SERVICES (CONT..)

Docker Service CLI:

- **Create a service**
  - docker service create [**image**]
- **List Services**
  - docker service ls
- **List the task (replica) of a service**
  - docker service ps [**service name**]
- **Delete a service**
  - docker service rm [**service name**]

Reference Doc :

https://docs.docker.com/engine/reference/commandline/service/

CHAPTER

Docker Services

DOCKER SERVICES

Scaling a service:

- Scale up or scale down a service that's running across swarm cluster.
- Replica flag used to create replica of containers.
  - docker service create --name mynginx --replicas 3 -p 80:80 nginx

Two ways to scale:

1. docker service update
   - docker service update --replicas 5 mynginx
   - docker service update --replicas 5 --deatach=true mynginx
     - -- detach=true: Not to see progress of service
2. docker service scale
   - Scale multiple services at a time.
     - docker service scale mynginx=2 mybusybox=3

Reference Doc :
https://docs.docker.com/engine/reference/commandline/service_update/
https://docs.docker.com/engine/reference/commandline/service_scale/

DOCKER SERVICES (CONT..)

Resource Limitation:

Defining containers CPU and memory requirements.

- docker service update **--limit-cpu**=.5 **--reserve-cpu**=.25 **--limit-memory**=124m **--reserve-memory**=64m mynginx
  - Limit
    - ✓ The maximum value of resource that can be used by container.
  - Reservation
    - ✓ The amount of resource required to run the container

Template with "docker service create“:

Template is used to give dynamic values.

- Flags can be used:
  - --mount
  - --hostname
  - --env
- docker service create --name mynginx2 **--hostname**="{{.Node.ID}}-{{.Service.Name}} " nginx

Reference Doc :

https://docs.docker.com/config/containers/resource_constraints/

https://docs.docker.com/engine/reference/commandline/service_create/

CHAPTER

Replicated & Global Mode

REPLICATED AND GLOBAL MODE

Replicated mode:

- Default mode.
- Can scale the service using **--replicas .**
  - docker service create --name nynginx --replicas 2 -p 80:80 nginx

Global Mode:

- Can't scale the service.
- **- - replicas** flag can't be used.
  - docker service create myglobalnginx -p 8080:80 --mode global nginx

Can't change the mode of a service.

Reference Doc :
https://docs.docker.com/engine/swarm/services/

CHAPTER

Docker Swarm - Quorum

SWARM QUORUM

Key Points:

- Majority of manager nodes in a swarm.
- More than half of the manager nodes in a swarm.
- Better having odd number of managers in a swarm.

Reference Doc:

https://docs.docker.com/engine/swarm/raft/

https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

Fault Tolerance:

Managers (N)	Fault Tolerance (N-1)/2	Quorum/Majority (N/2)+1
1	0	1
2	0	2
3	1	2
4	1	3
5	2	3
6	2	4
7	3	4
8	3	5
9	4	5

DOCKER SWARM - QUORUM

Key points to remember:

- More manager nodes affect the performance of swarm.
- Immediately replace failed manager node.
- Distribute manager nodes across Availability Zone (AZ) for High Availability (HA).
- Take swarm backup.

High Availability:

Managers	Quorum/Majority	Availability Zones
3	2	1-1-1
5	3	2-2-1
7	4	3-2-2
9	5	3-3-3

Distribution of manager nodes across 3 Availability Zones.

Reference Doc : https://docs.docker.com/ee/ucp/admin/configure/join-nodes/

CHAPTER

Constraint and Label

CONSTRAINTS AND LABELS

Constraint and Label:

Used to control the placement of containers.

Example 1:

Run tasks only on worker nodes.

- docker service create --name mynginx_worker \
  --**constraint** **node.role==worker** \
  --replicas 3 \
  nginx

Example 2:

Running tasks on particular node

1. Label
   - docker node update **--label-add** mynode=node1 [Node name]
2. Constraint
   - docker service create --name mynginx_dc1 \
     --**constraint** **node.labels.mynode==node1** \
     --replicas 3 \
     nginx

Reference Doc : https://docs.docker.com/engine/swarm/manage-nodes/#add-or-remove-label-metadata

CONSTRAINTS AND LABELS

Example 3:

Spread the tasks evenly across all nodes having label as mynode.

- docker service create --name mynginx_spread \
  --placement-pref spread=node.label.mynode \
  --constraint node.role==worker
  --replicas 4 \
  nginx

Reference Doc : https://docs.docker.com/engine/swarm/services/#placement-constraints

CHAPTER

Introduction to Docker Compose

INTRODUCTION TO DOCKER COMPOSE

Docker Compose:

Can run multi-container application using different images.

Install Docker Compose:

- Step 1:
  - Download docker compose binary to /usr/local/bin/docker-compose.
    - ✓ **sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**
- Step 2:
  - Provide executable permission
    - ✓ **sudo chmod +x /usr/local/bin/docker-compose**
- Step 3:
  - Check the version
    - ✓ **docker-compose --version**

Reference Doc :

https://docs.docker.com/compose/install/

https://docs.docker.com/compose/

INTRODUCTION TO DOCKER COMPOSE (CONT…)

Sample Docker Compose file:

```yaml
version: "3"
services:
  web_nginx:
    image: nginx
    ports:
      - "80:80"
  busybox_untility:
    image: radial/busyboxplus:curl
    command: /bin/sh -c "while true; do sleep 10; done"
```

```
docker service create \
        --name web_nginx \
         -p 80:80 \
          nginx
```

Reference Doc : https://docs.docker.com/compose/compose-file/

INTRODUCTION TO DOCKER COMPOSE (CONT…)

Build an image:

```
vi docker-compose.yml
```

```yaml
version: "3"
services:
  nginx_custom:
    build:
      context: .
      dockerfile: customnginx.dockerfile
    image: mynginx
    ports:
      - "8080:80"
  db:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: test123
    volumes:
      - .:/var/lib/mysql
```

```
vi customnginx.dockerfile
```

```dockerfile
FROM ubuntu

LABEL description = "My image for nginx Web Server"
LABEL version = "V1.0"

RUN apt-get update -y && \
    apt-get install curl -y && \
    apt-get install nginx -y

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

INTRODUCTION TO DOCKER COMPOSE (CONT…)

Docker Compose Commands:

- **Create a compose**
  - docker-compose up -d
- **List containers created by compose**
  - docker-compose ps / docker container ls
- **Stop a compose**
  - docker-compose stop
- **Start a compose**
  - docker-compose start
- **Restart a compose**
  - docker-compose restart
- **Delete a compose**
  - docker-compose down

Reference Doc : https://docs.docker.com/compose/reference/

CHAPTER

Docker Stack – Part 1

DOCKER STACK – PART 1

Docker Stack:

Can run services across the swarm.

Docker Stack Commands:

- **Deploy a stack**
  - docker stack deploy -c [**compose file name.yml**] [**stack name**]
- **List stacks**
  - docker stack ls
- **To see services associated with the stack**
  - docker stack services [**stack name**]
- **To see on what nodes tasks are running**
  - docker stack ps [**stack name**]
- **To see logs of a service**
  - docker service logs [**stack name**]
- **To remove a stack**
  - docker stack rm [**stack name**]

Reference Doc : https://docs.docker.com/engine/reference/commandline/stack/

DOCKER STACK – PART 1

Example-1: Creating Replicas

```yaml
version: "3.8"
services:
  web_nginx:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      mode: replicated
      replicas: 3
  busybox_untility:
    image: radial/busyboxplus:curl
    command: /bin/sh -c "while true; do sleep 10; done"
```

Service - 1

Service - 2

```
docker service create --name web_nginx \
                      -p 8080:80 \
                      --replicas 3 \
                      nginx
```

Reference Doc : https://docs.docker.com/compose/compose-file/

DOCKER STACK – PART 1 (CONT…)

Example-2: Using constraints and labels in docker compose file.

```yaml
version: "3.8"
services:
  web_nginx:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      mode: replicated
      replicas: 4
      placement:
        constraints:
          - "node.role!=manager"
        preferences:
          - spread: node.labels.datacenter
  busybox_untility:
    image: radial/busyboxplus:curl
    command: /bin/sh -c "while true; do sleep 10; done"
```

```
docker node update \
        --label-add datacenter=dc_1 \
        <node name>
```

Docker node update: covered in the past lessons/section.

CHAPTER

Docker Stack – Part 2

DOCKER STACK – PART 2

Example-3: Resource limitations.

```yaml
version: "3.8"
services:
  web_nginx:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      mode: replicated
      replicas: 4
      placement:
        constraints:
          - "node.role!=manager"
        preferences:
          - spread: node.labels.datacenter
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
  busybox_utility:
    image: radial/busyboxplus:curl
    command: /bin/sh -c "while true; do sleep 10; done"
```

```
docker service update --name web_nginx \
                      --limit-cpu=0.5 \
                      --reserve-cpu=0.25 \
                      --limit-memory=50m \
                      --reserve-memory=20m \
                      nginx
```

Docker service update: Covered in the past chapters/section

DOCKER STACK – PART 2 (CONT…)

Example-4: Using volume option.

```yaml
version: "3.8"
services:
  web_nginx:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      mode: replicated
      replicas: 4
      placement:
        constraints:
          - "node.role!=manager"
        preferences:
          - spread: node.labels.datacenter
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
    volumes:
      - type: volume
        source: myvolume
        target: /data

  busybox_utility:
    image: radial/busyboxplus:curl
    command: /bin/sh -c "while true; do sleep 10; done"

volumes:
  myvolume:
```

Deploy the Stack:

```
[root@Devops4Beginners ~]# docker stack deploy -c example4.yml app4
```

Inspect the service to see volume details:

```
[root@Devops4Beginners ~]# docker service inspect app4_web_nginx
```

```
"Mounts": [
    {
        "Type": "volume",
        "Source": "app4_myvolume",
        "Target": "/data",
        "VolumeOptions": {
            "Labels": {
                "com.docker.stack.namespace": "app4"
            }
```

DOCKER STACK – PART 2 (CONT…)

Example-5: Container communication.

```yaml
version: "3.8"
services:
  web_nginx:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      mode: replicated
      replicas: 4
      placement:
        constraints:
          - "node.role!=manager"
        preferences:
          - spread: node.labels.datacenter
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
    volumes:
      - type: volume
        source: myvolume
        target: /data
  busybox_utility:
    image: radial/busyboxplus:curl
    command: /bin/sh -c "while true; do echo 'HelloDocker'; curl web_nginx:80; sleep 10; done"
volumes:
  myvolume:
```

CHAPTER

Introduction to Docker Networking

INTRODUCTION TO DOCKER NETWORKING

Container Network Model (CNM):

- The Docker networking architecture is built on a set of interfaces called the **Container Networking Model** (CNM).
- **libnetwork** is the networking component which implements the CNM.

Docker network drivers:

1. Bridge
2. Overlay
3. Host
4. None
5. MACVLAN
6. 3rd party network drivers

Reference doc:
https://docs.docker.com/network/
https://success.docker.com/article/networking

INTRODUCTION TO DOCKER NETWORKING (CONT..)

Building blocks of CNM:

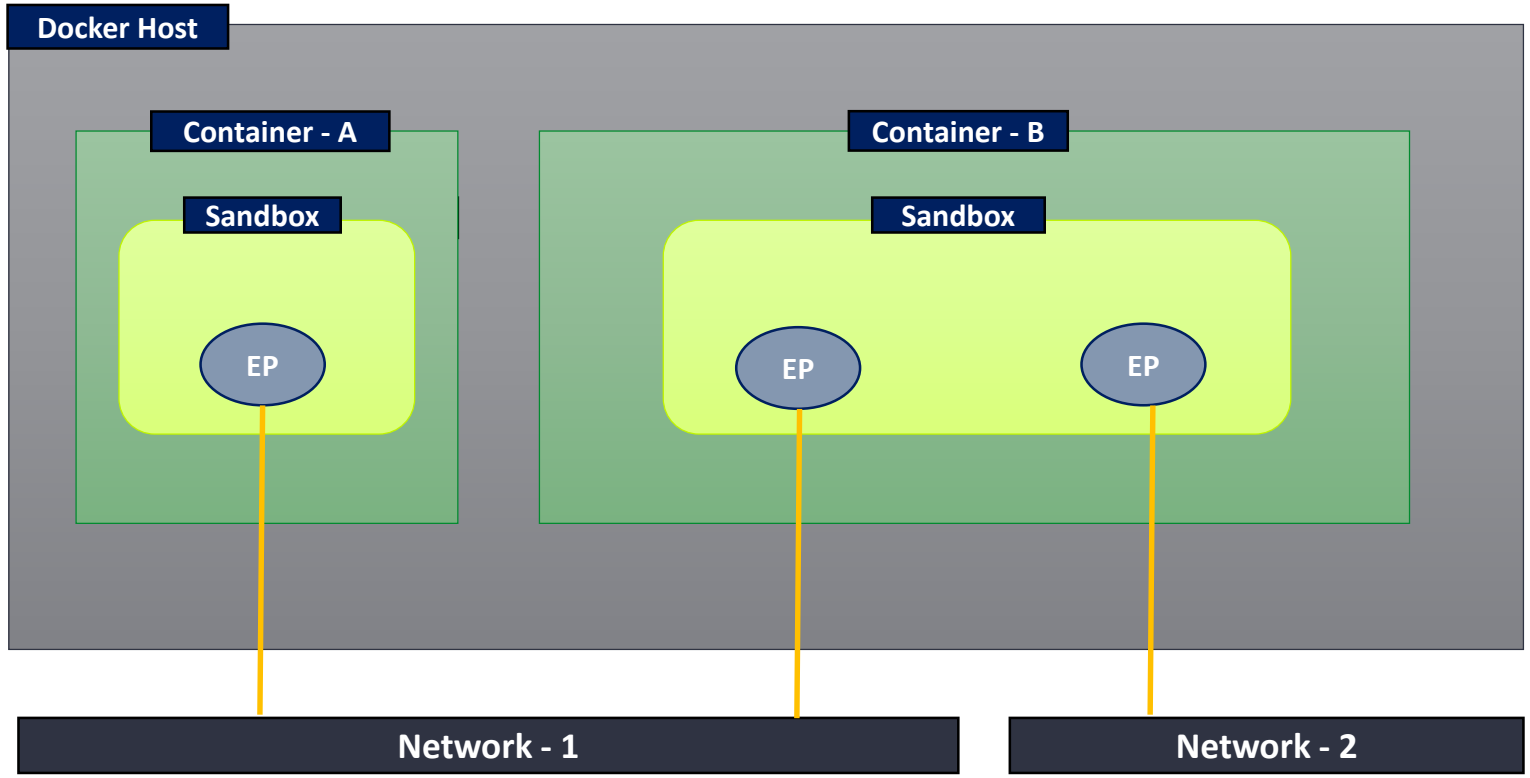


Diagram: Docker Networking. EP = Endpoints

1. **Sandbox**: Sandbox isolates the networking components of a single container such as network interfaces, ports, route tables and DNS.
2. **Endpoints**: Endpoints are virtual network interfaces and responsibility of endpoints is to connect the sandbox to a network.
3. **Networks**: Network is a collection of endpoints.

CHAPTER

Docker Networking Commands

DOCKER NETWORKING COMMANDS

Docker Networking Commands:

- **List Networks**
  - docker network ls
- **Create a network**
  - docker network create [**Network Name**]
- **Inspect a network**
  - docker network inspect [**Network Name**]
- **Connect a container to a network**
  - docker network connect [**Network Name**] [**Container Name**]
- **Disconnect a container from a network**
  - docker network disconnect [**Network Name]** [**Container Name]**

Reference doc: https://docs.docker.com/engine/reference/commandline/network/

DOCKER NETWORKING COMMANDS

Docker Networking Commands (Contd..):

- **Create a subnet and gateway**
  - docker network create --subnet 10.1.0.0/24 --gateway 10.1.0.1 [**Network Name**]
- **Assign a specific IP to a container**
  - docker container run -d --name [**Container Name**] \
    --ip [**IP Address**] \
    --network [**Network Name**] \
    nginx
- **Remove a network**
  - docker network rm [**Network Name**]
- **Remove unused networks**
  - docker network prune

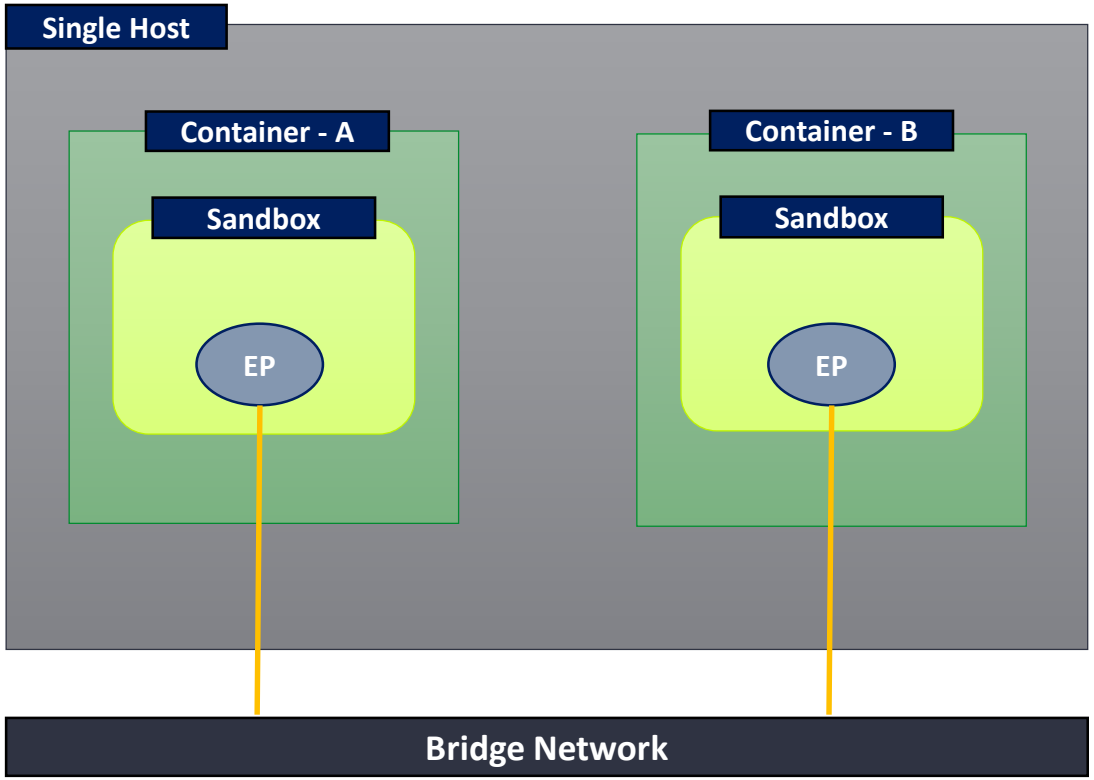
Reference doc: https://docs.docker.com/engine/reference/commandline/network/

CHAPTER

Bridge Network

BRIDGE NETWORK

Docker Bridge Network :



Docker Bridge Network:
Default network driver for containers running on a single host. (Not on Swarm).

Create a bridge network:
docker network create **--driver** bridge [**Network Name**]

(OR)

docker network create [**Network Name**]

Reference Doc: https://docs.docker.com/network/bridge/

CHAPTER

Docker's Embedded DNS

EMBEDDED DNS

Embedded DNS:

- Domain Name System (DNS).
- Name of container or services are mapped back to their actual IP address.
- Containers can communicate to each other using container name or service name, or network alias.

Commands:

- docker network create mynetwork
- docker container run -d --name mynginx --network mynetwork **--network-alias** mynetworkalias nginx
- docker container run -d --name mybusybox --network mynetwork radial/busyboxplus:curl sleep 1000
- docker exec -it mybusybox /bin/sh
  - curl mynginx:80
  - curl mynetworkalias:80

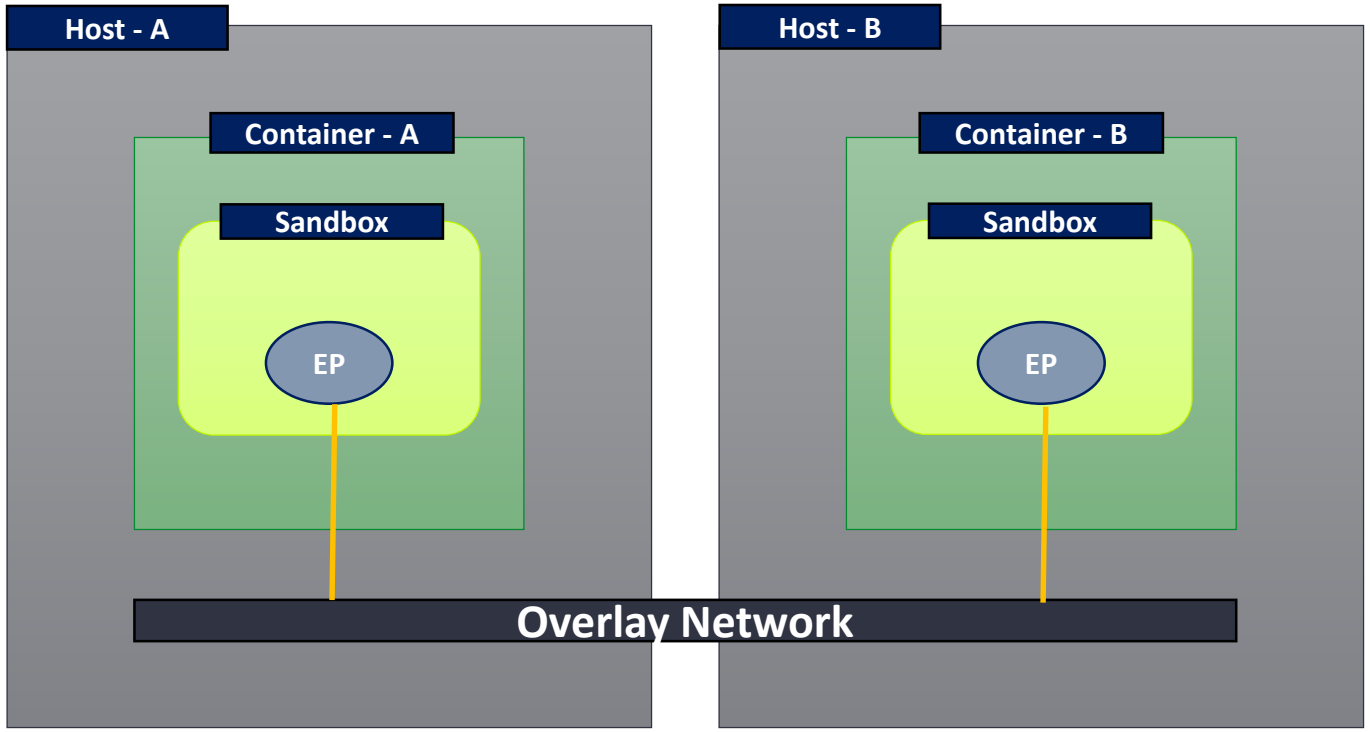
Reference Doc: https://docs.docker.com/engine/reference/run/#network-settings

CHAPTER

Overlay Network

OVERLAY NETWORK

Overlay Network:



Overlay Network:

- Overlay network allows containers running on same or different nodes (Multiple Hosts) to communicate with each other.
- Ingress is the default overlay network.
- Use flag **--driver=overlay** to create custom overlay network.

OVERLAY NETWORK (CONT..)

Commands:

- **Create a overlay network**
  - docker network create --driver overlay [**Network Name**]
  - docker network create --driver overlay --attachable [**Network Name**]
- **Create services with custom network**
  - docker service create -d --name mynginx --network [**Network Name**] --replicas 3 -p 80:80 nginx

Reference Doc:

https://docs.docker.com/network/overlay/

https://docs.docker.com/engine/reference/commandline/network_create/

CHAPTER

Host Network

HOST NETWORK

Host Network driver:

- No sandbox. No network component isolation.
- Uses Host's network infrastructure.
- Can not reuse the port.
- **Create a Host network:**
  - docker container run -d --name mynginx **--network host** nginx

Reference Doc:

https://docs.docker.com/network/host/

https://docs.docker.com/network/network-tutorial-host/

CHAPTER

None Network

NONE NETWORK

None Network:

- No Networking.
- Container is isolated from other container and also from host.
- **Create a none network:**
  - docker container run -d --name mynginxnone **--network none** -p 8080:80 nginx

Reference Doc:

https://docs.docker.com/network/none/

CHAPTER

Port Publishing Modes

PORT PUBLISHING MODES

Types of port publishing modes:

1. Ingress
2. Host

Ingress:

- The default mode.
- Publishes the port on all hosts i.e. all nodes of a swarm cluster. Routing-mesh.
- Create a service using ingress publishing port:
  - docker service create --name mynginx -p 8080:80 nginx

Host:

- Publishes the port on host where containers are running.
- Runs only one task of a service on the same node.
- Create a service using host publishing port:
  - docker service create --name mynginxhost -p mode=host,published=8081,target=80 nginx

Reference Doc: https://docs.docker.com/engine/swarm/services/#publish-a-services-ports-directly-on-the-swarm-node

CHAPTER

Introduction to Docker Security

INTRODUCTION TO DOCKER SECURITY

Docker Security:

- Uses both the Operating System (OS) and Docker native security features.

Linux Security Features:

- Namespaces
  - Process ID (pid)
  - Network (net)
  - Filesystem/mount (mnt)
  - InterProcess Communication (ipc)
  - User (user)
  - Unix Timesharing System (uts)
- Cgroups
  - CPU
  - RAM
- Seccomp

Reference Doc: https://docs.docker.com/get-started/overview/

INTRODUCTION TO DOCKER SECURITY

Some of Docker Security Features:

- Docker Content Trust (DCT)
- Docker Security Scanner
- Docker MTLS

Reference Doc:

https://docs.docker.com/engine/security

https://docs.docker.com/get-started/overview/#the-underlying-technology

https://docs.docker.com/ee/dtr/user/manage-images/scan-images-for-vulnerabilities/#the-docker-security-scan-process

https://docs.docker.com/engine/security/seccomp/

CHAPTER

Docker Security – Part 1

DOCKER SECURITY – PART 1

Secure Computing Mode (Seccomp):

- Using Secure Computing Mode (Seccomp) during container creation:
  - docker container run **--security-opt** seccomp=[Profile] Ubuntu
  - Example:
    - ✓ docker container run -it --name myubuntusec **--security-opt** seccomp=./default.json Ubuntu

Capabilities:

- Drop a capability:
  - docker container run **--cap-drop**=[Capability] [Image]
  - Example:
    - ✓ docker container run -it --name mybuntucapdrop --cap-drop=MKNOD ubuntu
- Add a capability:
  - docker container run **--cap-add**=[Capability] [Image]

Reference Doc:

https://docs.docker.com/engine/security/seccomp/#pass-a-profile-for-a-container

https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

INTRODUCTION TO DOCKER SECURITY

Docker Bench for Security:

- docker run -it --net host --pid host --userns host **--cap-add** audit_control \
  -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
  -v /etc:/etc:ro \
  -v /usr/bin/containerd:/usr/bin/containerd:ro \
  -v /usr/bin/runc:/usr/bin/runc:ro \
  -v /usr/lib/systemd:/usr/lib/systemd:ro \
  -v /var/lib:/var/lib:ro \
  -v /var/run/docker.sock:/var/run/docker.sock:ro \
  --label docker_bench_security \
  docker/docker-bench-security

Reference Doc:

https://github.com/moby/moby/blob/master/profiles/seccomp/default.json

https://github.com/docker/docker-bench-security

CHAPTER

Docker Content Trust

DOCKER CONTENT TRUST

Docker Content Trust (DCT):

- Verify integrity and publisher of an Image.
- Pull and run signed images.

Steps to set-up DCT:

Step 1:

- Log into the Docker Hub
  - docker login

Step 2:

- Generate a key (.pub)
  - docker trust key generate [**Docker hub username**]

Step 3:

- Add signer to an image repository:
  - docker trust signer add --key [**.pub**] [**Docker hub username**] [**repository**]

Reference Doc: https://docs.docker.com/engine/security/trust/content_trust/

DOCKER CONTENT TRUST (CONT..)

Step 4:

- Enable Docker Content Trust (DCT)
  - export DOCKER_CONTENT_TRUST=1

Step 5:

- Sign and push image to registry
  - docker trust sign [**Image**]:[**Tag**]

Disable Docker Content Trust (DCT):

- export DOCKER_CONTENT_TRUST=0

Logout of Docker hub:

- docker logout

Reference Doc:

https://docs.docker.com/engine/reference/commandline/trust_key_generate/

https://docs.docker.com/engine/security/trust/trust_delegation/#adding-additional-signers

https://docs.docker.com/engine/reference/commandline/trust_sign/

CHAPTER

Docker MTLS and encrypted overlay network

DOCKER MTLS AND ENCRYPTED OVERLAY NETWORK

Mutually Authenticated Transport Layer Security (MTLS):

- Docker Swarm uses mutual Transport Layer Security (TLS) for communication and authentication between nodes.

To Create an encrypted overlay network:

- docker network create **--opt encrypted** --driver overlay [**Network Name**]

Reference Doc: https://docs.docker.com/network/overlay/

CHAPTER

Uninstall Docker Engine

UNINSTALL DOCKER ENGINE

Uninstall Docker Engine:

- sudo systemctl stop docker
- sudo apt-get remove -y docker-ce docker-ce-cli
- sudo apt-get update

Reference Doc:

https://docs.docker.com/engine/install/ubuntu

https://docs.docker.com/engine/install/ubuntu/#uninstall-old-versions

CHAPTER

Logging Drivers

LOGGING DRIVERS

Logging Drivers:

By default Docker uses **json-file** logging driver.

Supported Logging Drivers:

Driver	Description
`none`	No logs are available for the container and `docker logs` does not return any output.
`local`	Logs are stored in a custom format designed for minimal overhead.
`json-file`	The logs are formatted as JSON. The default logging driver for Docker.
`syslog`	Writes logging messages to the `syslog` facility. The `syslog` daemon must be running on the host machine.
`journald`	Writes log messages to `journald`. The `journald` daemon must be running on the host machine.
`gelf`	Writes log messages to a Graylog Extended Log Format (GELF) endpoint such as Graylog or Logstash.
`fluentd`	Writes log messages to `fluentd` (forward input). The `fluentd` daemon must be running on the host machine.
`awslogs`	Writes log messages to Amazon CloudWatch Logs.
`splunk`	Writes log messages to `splunk` using the HTTP Event Collector.
`etwlogs`	Writes log messages as Event Tracing for Windows (ETW) events. Only available on Windows platforms.
`gcplogs`	Writes log messages to Google Cloud Platform (GCP) Logging.
`logentries`	Writes log messages to Rapid7 Logentries.

Reference Doc : https://docs.docker.com/config/containers/logging/configure/

LOGGING DRIVERS (CONT..)

Check default Logging driver:

- docker info
- docker info | grep storage

Method -1 : Edit unit file (docker.service)

- Add **--storage-driver** flag
  - sudo vi /usr/lib/systemd/system/docker.service
  - ExecStart=/usr/bin/dockerd **--storage-driver** devicemapper
- Restart the docker
  - sudo systemctl daemon-reload
  - sudo systemctl restart docker

LOGGING DRIVER (CONTD..)

Method 2: Configuration file (daemon.json)

- Configure daemon file
  - sudo vi /etc/docker/daemon.json

```
1  {
2      "log-driver":"syslog"
3  }
```

- Restart Docker
  - sudo systemctl restart docker
  - sudo systemctl status docker

THANK YOU

DevOps4Beginners